SPECIFICATION

Electronic Version 1.2.8
Stylesheet Version 1.0

METHOD AND SYSTEM FOR EVENT COMMUNICATION ON A DISTRIBUTED SCANNER/WORKSTATION PLATFORM

Background of Invention

[0001] The field of the invention is medical imaging methods and systems. More particularly, the invention relates to a method and system that allows for reliable and seamless delivery of events in a distributed medical imaging scanner and workstation platform having multiple hosts using one or more languages.

[0002] During typical operation of a medical imaging system having a scanner/workstation platform, many events are generated and received, where events are self-describing data packets provided between different applications (high level programs). Traditionally, a scanner/workstation platform was implemented on a single host computer. Further, the applications running on the host were typically programmed in the C++ language or another C-type language. Because of the structural simplicity of such systems and the degree to which the various elements of such systems shared the same communication protocols, event delivery between applications in such systems could be performed easily and rapidly, without the use of any specialized communication techniques.

[0003] However, new generation scanner/workstation platforms have become increasingly complex in their structure and operation. Many of these scanner/workstation platforms employ multiple hosts, use multiple architectures (e.g.,

little endian/big endian) and use multiple languages such as C++ or JAVA (for implementing user interfaces). Further, as scanner/workstation platforms have become integrated with the internet, the scanner/workstation platforms have further begun to employ and/or interact with applets and other internet protocols.

[0004] As a result of this increasing complexity of scanner/workstation platforms, the communication of events between applications running on the scanner/workstation platforms also has become increasingly complicated. In particular, the programming of applications will desirably take into account all of the intricacies affecting data transfer between applications that result from the use of multiple hosts, multiple architectures, and multiple languages, including internet communications protocols. Although such programming of applications that takes into account all of these intricacies is possible, such programming is complex and costly, and the resulting operation of scanner/workstation platforms is often slow and at times unreliable. Thus, programming to allow event communication using traditional methods requires a great deal of software knowledge on the part of the software programmer, and is difficult for the non-serious programmer (e.g., researchers who utilize the medical imaging systems).

[0005] It would therefore be advantageous if a system could be developed that allowed the applications running on medical imaging system scanner/workstation platform(s) to easily and reliably communicate events between one another, even when the scanner/workstation platform(s) included multiple hosts or employed multiple architectures. It would further be advantageous if such a system also allowed for easy and reliable communication of events between applications even when the applications were programmed in different computer languages. It would additionally be advantageous if such a system allowed for easy and reliable communication of events even between applications and applets, or between the applications of the scanner/workstation platforms and the internet.

Summary of Invention

[0006] The present invention relates to a medical imaging system having at least one host, and further including a first application, a second application, and an event

name service. The first application is at the at least one host, and is a first event source. The second application is at the at least one host, and is a first event listener capable of receiving a first event provided from the first application. The event name service is in communication with the first and second applications. At least one of the first and second applications is capable of registering with the event name service.

[0007] The present invention further relates to, in a medical imaging system, a system for communicating information that includes a plurality of event source applications, a plurality of event listener applications, a hosting means, and an event name service at which the event listener applications can register. The event source applications are capable of providing events, as publishers, to event listener applications. The event listener applications are capable of receiving, as subscribers, at least some of the events from the event source applications, using an event filtering mechanism. The hosting means is for hosting at least one of the plurality of event source and event listener applications. The event source applications only provide the events to the event listener applications if the event listener applications have registered for the events with the event name service.

[0008] The present invention additionally relates to in a medical imaging system, a method of communicating information. The method includes providing a first application that is an event listener, a second application that is an event source, and an event name service, and registering at least one of the event listener and the event source with the event name service. The method further includes sending an event from the event source to the event listener when the at least one of the event listener and the event source is registered with the event name service.

[0009] The present invention additionally relates to a method of communicating information on a distributed medical imaging scanner/workstation platform. The method includes providing a first event source on a first host of the platform, providing a first event listener on a second host of the platform, and registering the first event source at an event name service. The method further includes providing a listing of the first event source and a second event source in response to a request, determining whether a desired event source capable of providing a desired event as

required by the first event listener is registered and, when the desired event source is registered, firing the desired event from the desired event source to the first event listener.

Brief Description of Drawings

[0010] Fig. 1 is a schematic block diagram showing the interaction of applications on an exemplary distributed scanner/workstation that are operating to communicate events;

[0011] Fig. 2 is a schematic block diagram showing libraries used by the system of Fig. 1;

[0012] Fig. 3 is a schematic block diagram showing interfaces and objects within a DEMIDL C++ library of Fig. 2; and

[0013] Figs. 4-6 are schematic block diagrams showing wrapper classes within a DEM C++ library of Fig. 2; and

[0014] Figs. 7-10 are interaction diagrams showing the operation of APIs that allow communication of events between the applications of Fig. 1.

Detailed Description

[0015] Referring to Fig. 1, a block diagram is provided showing schematically how applications of a distributed medical imaging scanner/workstation platform 2 communicate events among one another. As shown, an application that is a source of events, namely, an event source 10, runs on a first host 22. The event source 10 can provide one or more events to an application that listens to events, namely, an event listener 20, shown to be running on a second host 24. Additionally, each of the event source 10 and the event listener 20 are in communication with a centralized location for discovering event services, namely, an event name service 30. The event name service 30 can exist on the first host 22, the second host 24, or elsewhere.

[0016] Fig. 1 is meant to be generally representative of any pair of applications running on a given medical imaging system. As shown, the two applications can be applications that are hosted by two different hosts, which can exist on either a single medical imaging system device, or on two separate medical imaging devices. In alternate embodiments, the two applications can exist on a single host. In further

alternate embodiments, the first application (i.e., the event source 10) is hosted by a medical device that is in communication with the second application of a second device by way of the internet. For generality, the scanner/workstation platform 2 of Fig. 1 is shown to have additional hosts 26, 28 on which other applications can exist and communicate with one another.

[0017] The two application programs that are communicating can also be operating on hosts that use different or multiple architectures (e.g., little Endian/big Endian), or can be hosted by hosts using a single architecture. Each of the event source 10 and the event listener 20 can be operating by way of any of a number of languages, such as C++ or Java. If the event source 10 is located on a different medical imaging device than the event listener 20, those two devices can be coupled by any one of a number of communication links, including the internet. The scanner/workstation platform 2 can include medical imaging devices of a variety of types, for example, magnetic resonance imaging (MRI) systems, CT systems, positron emission tomography (PET) scanner systems, x-ray scanners and nuclear imaging scanners.

[0018] The one or more events that are communicated between the event source 10 and the event listener 20 can be any sets of data that provide both substantive data and self-describing data. That is, the information being provided as an event should include information indicating how that information is to be operated upon. The events can range from simple strings (e.g., "hello") to complex arrays of values as well as to data having a variety of other formats.

[0019] One example of the first host 22 for the event source 10 would be a medical imaging scanner that has obtained a scanned image and has stored it on its database after a patient was scanned. In such case, an application of that medical imaging scanner might receive a request from an application running on another device, such as a printer, to provide the scanned information. In such case, if the information was transmitted from the medical imaging scanner to the printer, the event source 10 would be an application on the medical imaging scanner and the event listener 20 would be an application program on the printer.

[0020] As shown by Fig. 1, the event name service 30 is in communication with both the

event source 10 and the event listener 20. The event listener 20 is able to access information from the event name service 30 in the form of a list of all event sources of the scanner/workstation platform 2, which often will have multiple event sources operating simultaneously. The event name service 30 is further able to identify the event source or sources that could have generated any given particular event. The event name service 30 also acts as a surrogate for event sources that have not yet come on line. In such case where an event source has just come on line after being off line, the event name service 30 ceases to act as a surrogate. The event name service 30 operates as a single application that couples all of the hosts that are hosting the different applications that are potentially event sources or event listeners.

[0021] The relationship between the event source 10 and the event listener 20 occurs in accordance with a subscriber-publisher model by way of a distributed call back technique. At any given time, any given application can act as a publisher (that is, an event source) or as a subscriber (that is, an event listener). Applications that are the publishers (event sources) implement Application Program Interfaces (APIs) that are well-known to those skilled in the art, and that allow subscribers (event listeners) to register or unregister with the publishers. Further, the publishers employ well-known APIs to fire events in a multi-cast mode or to a group of subscribers that satisfy a filter criterion that is determined by the event name service 30. The subscribers for event listeners 20 implement well-known APIs that are invoked when events are being fired. The subscribers can further set the filters at the event name service 30, which indicate the kinds of events that the subscribers are interested in listening to or receiving. The filters can be simple restrictive filters that only allow a particular one of the events to be provided to a given subscriber, or be more complicated and allow certain combinations of events to be provided to a subscriber. The subscribers and publishers across the network can be located by the use of well-known service locator programs.

[0022] Turning to Fig. 2, a package diagram of a software framework used by the applications for communicating events on the distributed scanner/workstation platform 2 is provided. The language used by and among the various applications is typically the CORBA language, which is a distributed communication standard well-

known in the art. However, in alternate embodiments, other languages can be used. As shown in Fig. 2, the package diagram includes a DEM C++ library 40, which provides convenient wrapper methods and APIs that are in C++, which implement application level interfaces. The DEM C++ library 40 includes a DEMJNI C++ library 50 and a DEMjar Java library 60, which are the generated packages. The DEMJNI C++ library 50 contains cxxwrap generated C++ interfaces that are used by Java applications. The DEMjar library 60 contains Java APIs.

[0023] In the embodiment shown, each of the DEMJNI C++ library 50 and the DEMjar Java library 60 are in the Java native interface (JNI) format, which allows translation between the Java language and the C++ language. The DEM C++ library 40 further includes a DEMIDL C++ library 70, which implements the application level interfaces that provide the communication mechanisms for communication between different applications. The DEMIDL C++ library 70 defines the basic set of IDL interfaces, which are in the CORBA distributed communication standard. Although the package diagram of Fig. 2 includes C++ libraries and Java libraries and is configured for operation in accordance with the CORBA standard, in alternate embodiments, the particular languages and communication standards can vary.

[0024] Turning to Fig. 3, an exemplary DEMIDL package 170 that could be found within the DEMIDL C++ library 70 is provided. The DEMIDL package 170 is shown to include three interfaces, an ISource interface 110, an IListener interface 120 and an IName Service interface 130. The ISource interface 110 is the interface that all event sources should implement, either directly or by way of an object that is included within the event source. The IListener interface 120 is the interface that all event listeners should implement. The IName Service interface 130 is the interface that provides name lookup for the event sources and event listeners.

[0025]

Also included within the DEMIDL package is a FilterRep class 140, a ListenerRep class 150 and an EventRep class 160. Each of the FilterRep class 140 and the ListenerRep class 150 are used by the ISource interface 110. The FilterRep class 140 enables the event listener 20 to get called only when a desired type of event occurs. The FilterRep class 140 can be used as a single class or can be subclassed. The

ListenerRep class 150 is a callback object that allows the event source 10 to fire an event to the event listener 20. The EventRep class 160 is the base event that is delivered to the event listener 20 from the event source 10, and is utilized by the IListener interface 120. The ListenerRep class 150 is also in communication with the IListener interface 120, so that the ISource interface 110 can determine when to fire an event to the event listener 20 from the event source 10. The IName service interface 130 is in communication with the ISource interface 110, and the IName Service interface supports the number of APIs as shown.

[0026] Turning to Figs. 4-6, three sections 240a, 240b and 240c of an exemplary DEM package 240 are provided. The DEM package 240 corresponds to, and could be found or contained in the DEM C++ library 40. As shown, the DEM package 240 includes three classes: a TerraSourceUtil class 210 shown in section 240a in Fig. 4; a TerraListenerUtil class 230 shown in section 240b in Fig. 5; and a TerraNameServiceUtil class 260 shown in section 240c in Fig. 6. The TerraSourceUtil class 210 is a wrapper class that defines the APIs that are to be implemented for any application that is an event source, such as event source 10. The TerraSourceUtil class 210 provides a default implementation of these APIs, which can vary depending upon the embodiment. Additionally, the TerraSourceUtil class 210 hides all CORBA details and provides a simple and easy-to-use interface.

[0027] The TerraListenerUtil class 230 is a wrapper class that provides a mechanism for an event listener of a particular event (such as the event listener 20) to obtain or receive the event. The TerraListenerUtil class 230 employs a pointer (in this case, called "EventDispatcher") to allow the event listener 20 to get to the contents of the event. With respect to the TerraNameServiceUtil class 260, this is a singleton class that provides possible source references to event listeners, and is the centralized name server. The various event sources such as event source 10 are expected to register with the TerraNameServiceUtil class 260. Also, the TerraNameServiceUtil class 260 can register event listeners, such as event listener 20, that are waiting for particular event sources, before the event sources come up.

[0028]

As shown in Figs.the TerraSourceUtil class 210 includes an ISourceImpl subclass

220, and the TerraListenerUtil class 230 includes a TerraEventUtil subclass 250 and an IListenerImpl subclass 280. As shown in Fig. 6, the TerraNameServiceUtil class 260 includes a NameServiceUtil subclass 270. Every event listener 20 includes the TerraListenerUtil class 230 and the subclasses 250 and 280; every event source 210 includes the TerraSourceUtil class 210 and the ISourceImpl subclass 220, and every event name service 30 includes the TerraNameServiceUtil class 260 and the NameServiceUtil subclass 270. The DEMJNI C++ library 50 and the DEMjar Java library 60 include similar classes and subclasses that differ from those of Figs. 4-6 for the most part only insofar as the class and subclass information in the DEM package 240 of Figs. 4-6 has been wrapped by the cxxwrap program.

[0029] Referring to Fig. 7, an interaction diagram shows an exemplary interaction between the TerraSourceUtil class 210 of the event source 10 and the TerraListenerUtil class 230 of the event listener 20 during the firing of an event from the event source to the event listener. As shown, the interaction includes a first step 310, at which the TerraSourceUtil class 210 determines which event listeners are awaiting for an event, and a second step 320, at which the TerraSourceUtil class 210 fires the particular event to the TerraListenerUtil class 230 of the appropriate listeners, which in this case include the event listener 20. Turning to Fig. 8, an interaction diagram shows exemplary interactions between the TerraNameServiceUtil 260 and the TerraSourceUtil 210 of the event name service 30 and the event source 10, respectively, during the registering and unregistering of the event source with the event name service. As shown, the TerraSourceUtil class 210 can send either a register service command to the TerraNameServiceUtil class 260 at step 330, or an unregister service command to the TerraNameServiceUtil class at step 340, which respectively causes either the registering or unregistering of the event source 10 at the event name service 30, respectively.

[0030] Referring to Fig. 9, an interaction diagram shows how the TerraListenerUtil class 230 of the event listener 20 can contact the TerraNameServiceUtil class 210 of the event name service 30, in order to send information that the event listener is interested in receiving a particular type of event. To do this, a list service command is provided from the TerraListenerUtil class 230 to the TerraNameServiceUtil class 210 at

step 350.

[0031] Referring to Fig. 10, an interaction diagram shows how the event listener 20 is able to register for receiving an event even when an event source that is capable of providing the event is not yet present. In this case, the event name service 30 acts as surrogate or proxy for the desired event source, and transfers the registration once the event source comes on line. As shown, the TerraListenerUtil class 230 provides an add listener command at step 360 to the TerraNameServiceUtil 260 of the event name service 30. At step 370, the event name service determines if a desired event source for providing the desired event currently exists or is on line. Once the desired event source comes on line, the TerraNameServiceUtil class 260 provides an add listener command at step 380 to the TerraSourceUtil class 210 of the event source (e.g., the event source 10).

[0032] The embodiments shown in Figs. 1-10 are meant to be exemplary of a variety of different embodiments in which one or more event listeners are in communication with one or more event sources, by way of one or more event name services. The embodiments shown allow applications corresponding to the event sources and event listeners that are written in either C++ or Java to operate by way of the same programming interface. Further, the embodiments shown scale easily for applications that are on multiple hosts and are written in multiple languages or utilize applet communications.

[0033] While the foregoing specification illustrates and describes the preferred embodiments of this invention, it is to be understood that the invention is not limited to the precise construction herein disclosed. The invention can be embodied in other specific forms without departing from the spirit or essential attributes of the invention. Accordingly, reference should be made to the following claims, rather than to the foregoing specification, as indicating the scope of the invention.